Measuring Software Engineering

Why do we measure software engineering ?

There are numerous reasons as to why an individual or firm may set out to measure the software engineering process. Software metrics are used in order to better manage the software development process and the main goal of such metrics is to obtain measurable data which is then used to better understand what processes are taking place and to assess performance. With this knowledge, changes can be made during development in order to obtain desired results.

There are many advantages and disadvantages to using metrics, and whether or not they are implemented depend on the nature on the software being developed. Some of the benefits of metrics are they can help save development effort, time and money. They can also provide quality assurance as any problems can be identified early on, they can help manage the resources required and can provide an overview of progress made towards completion. (Fernando, Wijayarathne, Fernando, Mendis, & Guruge, 2014) Above all, metrics can be used to assist in the decision making of those in managerial positions who oversee the development of software. (Fenton & Neil, 1999)

However, there are also drawbacks to measuring these process. Unlike with most engineering disciplines, software engineers don't place a high emphasis on measuring and analysing the ongoing processes using metrics. This is primarily due to the high cost of time and effort that occurs when attempting to collect data. Since the collection of data does not produce immediate results, it is often overlooked. Furthermore, much data collection is done manually by software engineers, the accuracy of said data may be questionable and this further increases the workload on software developers.

Indeed in the past, separate teams of software developers were assigned to a "Software Process Group" whose sole purpose would be the collection and analysation of data. This was done as to not increase developer workload which may stifle productivity due to context switching. (P. M. Johnson, 2003) Although now there are tools which automate the collection of data such as PROM and HackyStat, data collection still remains a resource intensive task which still relies heavily on human effort.

The issue of standardization also exists as there are no consolidated metrics that maybe applicable to all software development processes, which further contributes to the difficulty of finding automated processes to collect and analyse data. This can be attributed to the changing the technological dependencies of software development projects and the exact processes, sequence of tasks and activities that take place during development cannot be known prior to commencing. (Grambow, Oberhauser, & and Reichert, 2013)

Properties of metrics

There are many different kinds of metrics in use such as product metrics, process metrics, project metrics, quality metrics and testing metrics. A few key points when implementing metrics should be to avoid creating a burden on the software developer as they will be less incentivised to carry out such measurement frequently. It is beneficial to link metrics to certain goals and focus on overall trends in the software development process as opposed to objective numbers. Time spent measuring could also be set to small time frames as to not hinder productivity and metrics that are not working should be replaced quickly with ones that produce verifiable results.

Metrics in general should be (Altvater, 2017):

- Simple and computable
- Consistent and unambiguous (objective)
- Use consistent units of measurement
- Independent of programming languages software products
- Easy to calibrate and adaptable
- Easy and cost-effective to obtain
- Able to be validated for accuracy and reliability
- Relevant to the development of high-quality

Types of Metrics

Source Lines of Code/Lines of code

LOC or SLOC is a measure of the size of a software program based on the number of lines of code. It is used to estimate the effort required to develop a program, the program productivity and the maintainability of the software following development. (Bhatt, Tarey, & Patel, 2012) It has its advantages and disadvantages which should be taken into consideration as LOC is used as an input in other metrics such as the defect density metric.

LOC is a physical entity and therefore it is a metric that can be visualised and easily automated once it is given parameters as to what can counted as a LOC for a certain programming language. However, given that parameters for structure and syntax for a line of code varies from language to language, it can be difficult to scale up such a metric to many different programming languages without considerable human effort.

LOC is does not give a complete measure of the effort that went into developing a program as it only accounts for the coding aspect of software development which only accounts for a fraction of the total development process. Furthermore, the performance and effort of highly skilled programmers may be understated due to the fact that high level programmers can achieve the same functionality as a lower level programmer with few lines of code. It may provide the wrong incentive for programmers to write excess and unnecessary code in order to inflate the line count. (Bhatt, Tarey, & Patel, 2012)

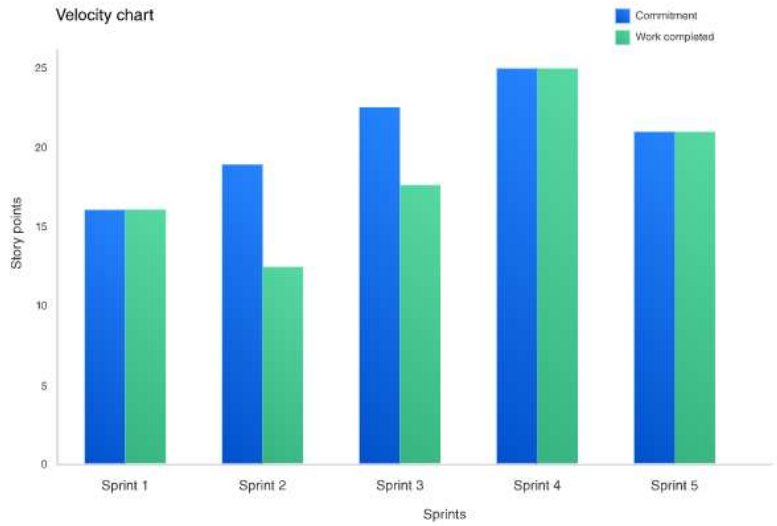
The biggest drawback to the LOC measurement is the lack of standardization in what counts as a line of code. Generally there are two types of LOC, physical LOC and logical LOC. Physical lines count all lines of codes including comments while logical lines count only executable statements. There are numerous variations of this principle within and between programming languages. This problem is only made worse with the introduction of newer programming languages each year.

In the past, LOC was primarily used to measure the size of programs written in assembly language in which each line of code represented a statement. With the introduction of high level and abstract programming languages this one-to-one correspondence was no longer applicable and account for the variations in counting LOCs. (Kan H, 2002)

Despite its flaws, LOC is still considered to be a successful metric due to its widespread use in the software industry. This is primarily due to simplicity and the ability to visualise the results produced (Fenton & Neil, 1999). As a result, the metric is still used as a measure of software quality, worker productivity and it also acts as an input in metrics such as defects per KLOC (thousand lines of code), cost per KLOC and errors per KLOC.

Team Velocity

Team velocity is a measure of how much work a team completes in one sprint or iteration. It is a measure of how fast a team is working and it is measured in story points or hours. It can act as a measure of productivity; however it should never be used to compare the productivity of two teams as different groups can interpret progress in different ways. For example, one team might award points only to any new functionality added while another may award themselves points for bug fixes as well. Through this metric, management can estimate a completion date for the software program and observe progress being made. Velocity should initially increase as the team members begin to work together and optimise the work process and can ensure consistent productivity if they have a consistent velocity. A decrease in the velocity can also signal inefficiency or difficulty faced by the team. An overzealous goal may also lead to a decrease in velocity and so will any unforeseen difficulties, so a team must always set realistic expectations and deadlines as to not rush through development and compromise the quality of the software. (Radigan, n.d.)

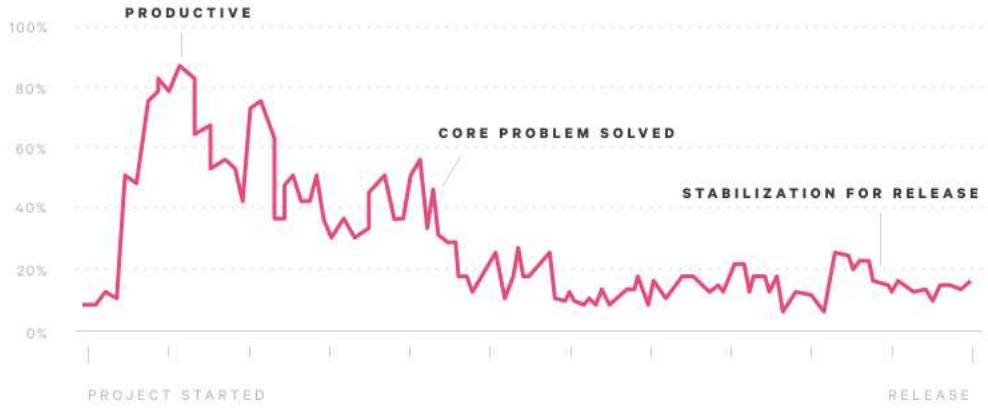


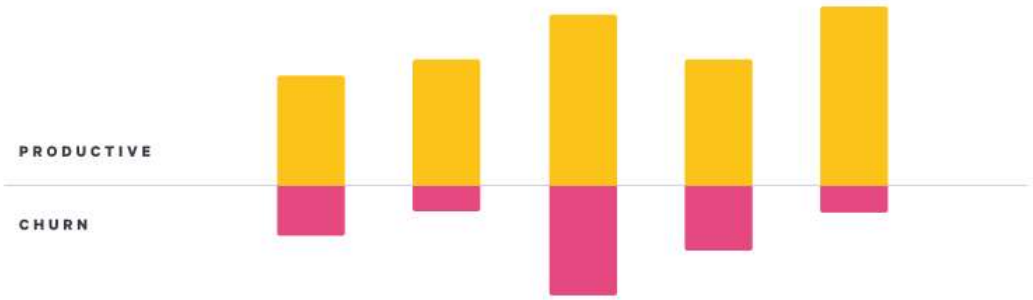
Code Churn

Code Churn is a metric that shows the lines of code that have been rewritten or changed shorty after being written. Code churn is a normal aspect of the software development process and can vary from team to team. It is not necessarily a bad indicator when the levels of code churn are at normal levels.

Typically high levels of code churn can be observed at the beginning of a project when there is a large amount of prototyping taking place and towards the end when the program is being polished. However, when an unusually high level of code churn is observed from a programmer, it can be a sign that they are facing difficulty with their task and in this way help can provided to them. However, it may also be an indicator of unproductivity as if existing code is simply being re-written without any addition of value then this would be a waste of time as this effort is better spent elsewhere. (Pluralsight, 2019)

Furthermore, it has been shown that frequent code changes accelerate the pace of code erosion. The source code of software typically erodes over the software development process which increases the cost of maintenance. Frequently changed code has been shown to have a negative effect on software maintainability than less frequently maintained code. This means that higher levels of code churn typically lead to higher maintainability costs and also typically lead to a higher number of defects. (Farago, Hegedys, & Ferenc, 2015)





Function Point Analysis

Function points are a unit of measurement that quantifies the business functionality provided by the product. It measures functionality from the users point of view i.e. on the basis of what the user requests and receives in return. The cost in (in dollars or hours) of a single unit is calculated from past projects. Function points were defined in 1979 by Allan Albrecht of IBM.

This measurement works by analysing the functional user requirements of the software program and categorising them into one of five types : outputs, inquiries, inputs, internal files, and external interfaces. Once a function in the program is analysed and placed into one of the mentioned categories, it is assigned a number of function points. Each of the functional user requirements are linked with an end user business function. For example, a result displayed by a function can be an mapped to an output and a query from the user can be mapped to an inquiry. (Parlati, Larenza, & Caronia, 2011)

The advantages of such a metric is that the software development process can be measured independently of the technology that was used to implement the software so it can be useful for comparing software written in different programming languages. Function point analysis also works to simply the complexity of software as it provides the user with the functionality that can be provided, and the user can see if the software matches their requirements. This metric can also show how many units are in a software product thus allowing developers to estimate the size of the software and thus the cost of development and maintenance. (GeeksforGeeks, 2019)

Computational Platforms

Pluralsight

Pluralsight is a platform available to software development teams and allows for collaboration and co-ordination. However the aspect of Pluralsight that is relevant to measuring the software development process is "Flow" ,which was formerly known as GitPrime. Flow makes use of online code repositories such GitHub and Bitbucket and can be used to collect and measure data regarding the workflow of a team. One can gain insights such as the commits made by team members, any open pull requests and any bottlenecks within the team. Using historical data, Flow can also provide information regarding trends within the development team and can compare such trends with industry benchmarks. The visualisations provided can also show how management decisions made can affect productivity and workflow within the team, which assists management in making more informed decisions.

Hackystat

Hackystat is the third generation of PSP data collection tools developed at University of Hawaii. The framework allows developers to collect process data and analyse them automatically (Sillitti, Janes, Succi, & Tullio, 2003). The main benefit of Hackystat from its predecessors, like PSP and LEAP, is its automation of data collection via sensors. These sensors are attached to development tools and collect metrics such as program size, activity and defects and sends them to a centralised server to be analysed. The metric data is analysed every 5 mins on the server side, and a representation of the programmer's activity in 5 mins intervals is represented in what is known as the "Daily Diary" which can act as input for larger visualisations for trends such as effort given per day. (P. M. Johnson, 2003)

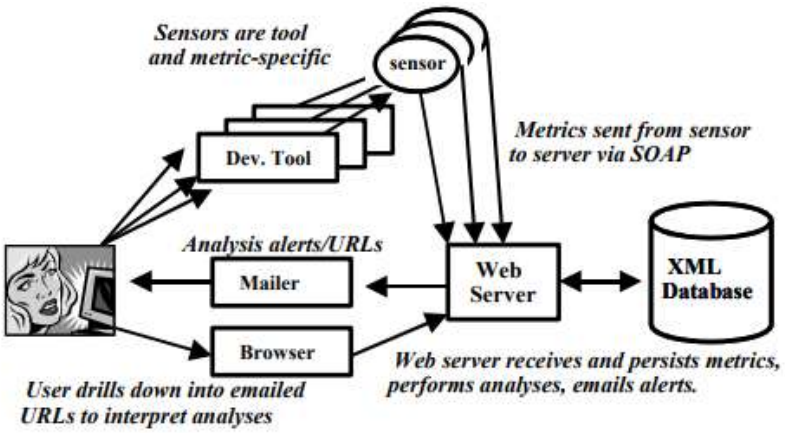


Figure 2. The basic Hackystat architecture and information flow

Furthermore, there exists features in Hackystat such as the ability to be notified via email if certain thresholds are exceeded by the developer such as the size or complexity of the program and this means that the developer is not required to manually switch to Hackystat to view any pending problems. This level of automation helps to increase the acceptance rate of Hackystat from its predecessors, reduce the overhead on developers and eliminate the problem of context switching allowing for greater productivity.

There are issues relating to the challenge of widespread adoption by development tools as not all tools can be implemented with Hackystat sensors. There are also privacy concerns given the detail to which information is collected and analysed, however, Hackystat provides solutions to this problem by having servers be available offline or locally on one's computer depending what level of control developers prefer over their data. (P. M. Johnson, 2003)

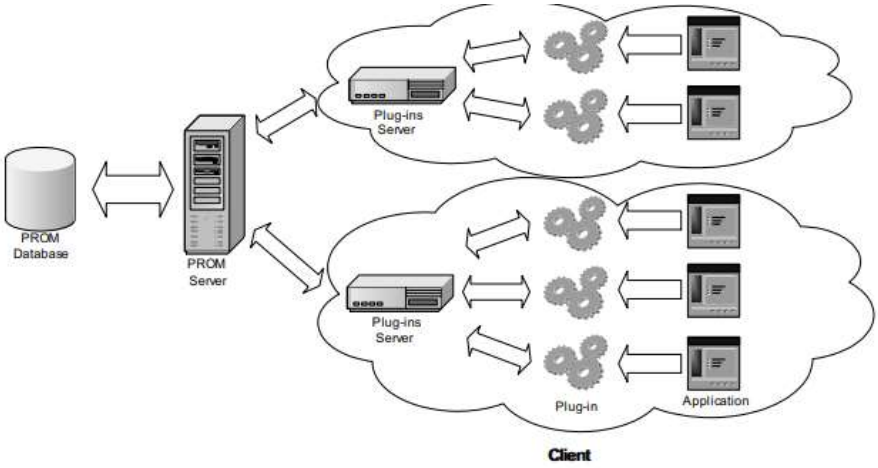
Hackystat primarily focuses on usage by individuals as opposed to the collective such as a large development team which allows for greater privacy, at the cost of limited usability to an organisation (Sillitti, Janes, Succi, & Tullio, 2003). However, from its lightweight implementation and automation of data collection, Hackystat has been a pioneer in the field of metric data collection.

PROM

PROM (PRO Metrics) is a tool for automated data acquisition and analysis that collect both code and process measures. The tool focuses on a comprehensive data acquisition and analysis to provide elements to improve products. The tool collects and analyses data at different levels of granularity: personal, workgroup and enterprise. (Sillitti, Janes, Succi, & Tullio, 2003) PROM sets out to collect all PSP metrics, procedural and Object-Oriented metrics but can also measure activities that aren't coding such as documentation. PROM differs from Hackystat because it is designed to collect and analyse data to be viewed by the individual and the larger organisation. However, it ensures developers' privacy is maintained by only displaying aggregate data on the whole software development as opposed to a specific individual.

PROM is also available for a wide range of IDEs and collects data through the use of plugins. Similarly to Hackystat, PROM allows it developers to work offline but it also allows multiple users to login and have their data measured and analysed.

This approach allows for pair programming and the sharing of data between users can also benefit developers in that they can obtain valuable feedback from their peers. Furthermore, in order to increase measurement, users can manually insert any data that is outside of the coding process through a web page and the accuracy of such data can be kept to a high standard so long as the manual data collection is focused and performed sparingly.



Algorithmic Approaches

Cyclomatic Complexity

Cyclomatic complexity is a metric used to measure the complexity of a program that is dependent on an algorithmic approach. It can be computed using a control flow graph of the code and measures the linearly independent lines of code within a program. Within the control flow graph, the nodes represent the tasks that are being processed and the edges represent the control flow between the tasks. A higher cyclomatic number means that the code is more complex, and an increase in complexity generally correlates with an increase in the number of defects present and makes it difficult to understand the code. Furthermore, it can be demonstrated that there exists a relationship between the complexity of a program and its cohesion, such that a lack of cohesion within a program generally leads to higher complexity. (Ikerionwu, 2010)

A benefit of calculating the cyclomatic complexity is that the number produced can provide an upper bound to the number of tests that can be carried out in order to ensure that all statements have been executed at least once. (Ikerionwu, 2010)

Once a control flow graph has been produced, the complexity can be calculated as follows :

Cyclomatic complexity = E - N + 2*P, where :

- E = number of edges in the flow graph.
- N = number of nodes in the flow graph.
- P = number of nodes that have exit points (TutorialsPoint, 2020)

The main benefits of measuring complexity using this algorithm is that it can be used to compare the complexity between different designs, and it is able to guide the testing process. Its simplicity also means that its relatively easy to implement into a software development process and can be used to measure the quality of a program.

The main drawbacks are that cyclomatic complexity is a measure of the control complexity as opposed to the data complexity of a program and the same weight is given to nested and non-nested loops despite nested conditional structures adding to complexity more than non-nested structures. (GeeksforGeeks, Cyclomatic Complexity, 2020)

Bayesian Belief Nets

A Bayesian Belief Net is a graphical network with an associated set of probability tables. The nodes represent uncertain variables and the arcs represent the causal/relevance relationships between the variables (Fenton & Neil, 1999). A BBN works in conditions of uncertainty when information for all metrics may not be available. With BBNs one can observe the dependencies between different variables and the impact that new information can have on the existing variables to predict future outcomes such as the reliability of code. (Neil & Fenton E, 1996)

BBNs are also provided with a probability table that shows the probability of each state that a variable could be in. A BBN can contain a mixture of the variables. If we already have information about program complexity, then we can use these values to predict the values of other variables. These values change constantly with the any new information added. The main benefit of a BBN is that it can compute the probability of every state of every variable regardless of the amount of evidence available . However, the lower the amount of evidence the higher the uncertainty of each remaining variable. (Fenton & Neil, 1999)

Another benefit of BBNs is the ability to visualise and manipulate complex models that might never be implemented using conventional methods. This ability can be used to simulate and compare the effect of different decisions made by management on different variables such as defect density and quality.

Fig. 1. A BBN that models the software defects insertion and detection process.

Halstead Complexity Measures

Halstead complexity measures are software metrics introduced by Maurice Howard Halstead in 1977. Halstead believed that a computer program is a type of algorithm that consists of collection of token that can either be defined as an operator or operand.

In this way, all of Halstead's metrics can be defined using these basic measures (javaTpoint, 2018) :

- n1 = count of unique operators.
- n2 = count of unique operands.
- N1 = count of total occurrences of operators.
- N2 = count of total occurrence of operands.

In terms of the total tokens used, the size of the program can be expressed as N = N1 + N2.

Through these values, a number of metrics can be calculated to better quantify the software development process.

Metric	Meaning	Mathematical Representation
n	Vocabulary	n1 + n2
N	Size	N1 + N2
V	Volume	Length * Log2 Vocabulary
D	Difficulty	(n1/2) * (N1/n2)
E	Efforts	Difficulty * Volume
B	Errors	Volume / 3000
T	Testing time	Time = Efforts / S, where S=18 seconds.

(TutorialsPoint, Software Design Complexity, 2020)

The numerous benefits of using the Halsted metrics include simplicity in calculation and its ability to measure the overall quality of the code. These metrics can be scaled up to be used in any programming language and can also be a good predictor of the maintenance effort required post release. However, problems arise when there is a problem distinguishing between operators and operands as there are no generic definitions of what an operator and operand are that will apply to every programming context (Al-Qutaish & Abran, 2005). Furthermore, a big drawback of this method is that it can only be applied to completed programs and cannot be used on works in progress.

Ethics

When measuring the software development process, one must consider if the data that is being collected and analysed to assess performance is done so ethically. This process must ensure that the privacy of the developer is respected, and only aggregate data is viewed by management. We have seen this type of implementation in platforms such as PROM but with tech companies coming under increased scrutiny as of late regarding privacy concerns, it is vital that a precedent is set in order to ensure the privacy of users and developers alike.

If a company was to utilise the software development metrics to assess the performance of employees, problems may arise.

A simple example would be using the LOC measure to assess the productivity of a team member. This may in fact stifle productivity as if an employee is incentivised to write more lines of code to resolve an issue as opposed to a shorter, more efficient method, human resources would be wasted. Furthermore, this would only serve to add complexity to the program and as famous computer scientist, Edsger Dijkstra said "Simplicity is the pre-requisite for reliability" .

If one were to assess employee performance via efficiency, this would also create a negative effect on the software quality. This would incentivise developers to rush through aspects of development to ensure a favourable measure of efficiency as opposed to creating better quality code. It is important that data is obtained from a variety of metrics and is only viewed as an aggregate in order to ensure accuracy and guarantee that it meets ethical standards.

Rank and Yank Management System

One controversial measure of employee assessment is the "rank and yank" system of management. This was system of assessment developed by Jack Welch, CEO of General Electric, in the 1980s and works by instructing managers to rank subordinates in a company to create a hierarchical structure. Those at the bottom will be let go from the company even if they aren't necessarily poor performers. This creates problems of good workers being let go in the hopes those left after each removal will be the cream of the crop. (Business.com, 2020)

However, this practise has been shown to demoralise employees and only works to reduce innovation. This is because employees don't want to take on new projects due to the risk that the project may fail and they will be dropped to a lower rank. Similarly, top-level employees will avoid working together on a project as they may be forced to compete with one another and inevitably a few will be reduced in rank. This practise had been implemented by companies like Yahoo and Microsoft with the latter abandoning this method as of late.

This kind of working environment serves to create an intense competitive atmosphere, that removes the prospect of a work life balance for the employees. This can be shown with the likes of many American tech firms, as despite implementing and promoting better policies, so long as their remains a surplus of young, educated workers ready to fill an open position, policy changes will make no difference in persuading people to take time off work. (Scheiber, 2015)

Even if employees take time off work, it is shown that many employees still continue to use messaging apps such as Slack for casual conversations and this keeps the employees in a pseudo working environment despite being off work and serves to create an implicit pressure to be available to colleagues at all times. (Scheiber, 2015)

There are numerous ethical concerns regarding comparing employees with one another due to the psychological toll it may take on the employees becoming demoralised and feeling inadequate when being ranked against their peers. It forces employees to only look out for themselves and causes a lack of team spirit. There is also a physical toll at work as many employees voluntarily ignore a work-life balance and push themselves to overwork in order to guarantee the safety of their current position.

As of late this system is being phased out for alternative measurements, with the exception of a few firms. However, it can also be argued that the a different, less biased form of employee ranking is implicitly used to compare employees to one another.

Bibliography

Al-Qutaish, R. E., & Abran, A. (2005). An Analysis of the Design and Definitions of Halstead's Metrics. *15th International Workshop on Software Measurement* , (pp. 337-352). Montreal, Canada.

Altvater, A. (2017, September 16). *What Are Software Metrics and How Can You Track Them?* Retrieved from Stackify: https://stackify.com/track-software-metrics/

Bhatt, K., Tarey, V., & Patel, P. (2012). *Analysis Of Source Lines Of Code(SLOC) Metric.* Ujjain: International Journal of Emerging Technology and Advanced Engineering.

Business.com. (2020, June 10). *The End of Rank and Yank: Management Practices Revisited*. Retrieved from Business.com: https://www.business.com/articles/the-end-of-rank-and-yank-management-practices-revisited/

Farago, C., Hegedys, P., & Ferenc, R. (2015). Cumulative Code Churn: Impact on Maintainability. *2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)* (pp. 1-2). Bremen, Germany: IEEE.

Fenton, N. E., & Neil, M. (1999). Software metrics: successes, failures and new directions. *The Journal of Systems and Software*, 149-157.

Fernando, W., Wijayarathne, D., Fernando, J., Mendis, M., & Guruge, I. (2014). *THE IMPORTANCE OF SOFTWARE METRICS: PERSPECTIVE OF A SOFTWARE DEVELOPMENT PROJECTS IN SRI LANKA.* Colombo: SAITM Research Symposium on Engineering Advancements 2014 .

GeeksforGeeks. (2019, May 21). *Software Engineering | Functional Point (FP) Analysis*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/software-engineering-functional-point-fp-analysis/

GeeksforGeeks. (2020, August 5). *Cyclomatic Complexity*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/cyclomatic-complexity/

Grambow, G., Oberhauser, R., & and Reichert, M. (2013). Automated Software Engineering Process Assessment: Supporting Diverse Models using an Ontology. *International Journal on Advances in Software*, 213-224.

Ikerionwu, C. (2010). *Cyclomatic Complexity As A Software Metric.* Owerri, Imo State, Nigeria: International Journal of Academic Research.

javaTpoint. (2018). *Halstead's Software Metrics*. Retrieved from javaTpoint: https://www.javatpoint.com/software-engineering-halsteads-software-metrics

Kan H, S. (2002). *Metrics and Models in Software Quality Engineering, 2nd Edition.* Addison-Wesley Professional.

Neil, M., & Fenton E, N. (1996). Predicting Software Quality using Bayesian Belief Networks. *Proceedings of 21st Annual Software Engineering WorkshopNASA/Goddard Space Flight Centre*, (pp. 3-5). London.

P. M. Johnson, H. K. (2003). Beyond the Personal Software Process : Metrics collection and analysis for the differently disciplined. *Proceedings of the 25th international Conference on Software Engineering, IEEE Computer Society.*

Parlati, G., Larenza, R., & Caronia, L. (2011). Measuring software 4 dummies. *PMI® Global Congress 2011.* Dallas, TX: Project Management Institute.

Pluralsight. (2019, November 2019). *Introduction to Code Churn*. Retrieved from Pluralsight: https://www.pluralsight.com/blog/tutorials/code-churn

Radigan, D. (n.d.). *Five agile metrics you won't hate*. Retrieved from Atlassian: https://www.atlassian.com/agile/project-management/metrics

Scheiber, N. (2015, August 17). Work Policies May Be Kinder, but Brutal Competition Isn't. *The New York TImes*.

Sillitti, A., Janes, A., Succi, G., & Tullio, V. (2003). Collecting, integrating and analyzing software metrics and personal software process data. *Proceedings of the 29th Euromicro Conference*, (pp. 336-342).

TutorialsPoint. (2020). *Software Design Complexity*. Retrieved from TutorialsPoint: https://www.tutorialspoint.com/software_engineering/software_design_complexity.htm

TutorialsPoint. (2020). *What is Cyclomatic Complexity?* Retrieved from TutorialsPoint: https://www.tutorialspoint.com/software_testing_dictionary/cyclomatic_complexity.htm